Exhibit L

Error control coding
An introduction

Peter Sweeney
Department of Electronic and Electrical Engineering
University of Surrey

Prentice Hall
New York London Toronto Sydney Tokyo Singapore

First published 1991 by
Prentice Hall International (UK) Ltd
66 Wood End, Hemel Hempstead
Hertfordshire HP2 4RG
A division of
Simon & Schuster International Group

Typeset in 10/12 pt Times by Columns Design and Production Services Ltd, Reading

Printed and bound in Great Britain
by Dotesios Printers Ltd, Trowbridge, Wiltshire.

Library of Congress Cataloging-in-Publication Data

Sweeney, Peter, 1950–
Error control coding : an introduction / by Peter Sweeney.
p. cm.
Includes bibliographical references and index.
ISBN (invalid) 0–13–248119–5
1. Coding theory. 2. Error-correcting codes (Information theory)
I. Title
TK5102.5.S958 1991
629.8'315–dc20 90–7522
CIP

British Library Cataloguing in Publication Data

Sweeney, Peter *1950–*
Error control coding : an introduction.
1. Error-correcting codes
I. Title
005.72

ISBN 0–13–284118–5
ISBN 0–13–284126–6 pbk

1 2 3 4 5 95 94 93 92 91

1.5 Example of block coding

In order to see more clearly some of the effects of error control coding, an example of a binary block code is presented in Table 1.1. Encoding will be done by looking up the codeword that corresponds to the information. After introducing errors, decoding will be performed by comparing the received sequence with every codeword to find the one that is the least distance from the received sequence. In the hard decision case this boils down to a simple count of discrepancies, known as Hamming distance. Suppose we wish to transmit the information 1100. From Table 1.1, the codeword is 1100011. Note that the codeword consists of the information followed by some extra bits; these bits have in fact been calculated from the information in a manner that will be explained in Chapter 2. This construction of information symbols followed by further calculated symbols is common in block codes, the code being said to be *systematic*.

Table 1.1 Example of a block code

Information	Code
0 0 0 0	0 0 0 0 0 0 0
1 0 0 0	1 0 0 0 1 1 0
0 1 0 0	0 1 0 0 1 0 1
1 1 0 0	1 1 0 0 0 1 1
0 0 1 0	0 0 1 0 0 1 1
1 0 1 0	1 0 1 0 1 0 1
0 1 1 0	0 1 1 0 1 1 0
1 1 1 0	1 1 1 0 0 0 0
0 0 0 1	0 0 0 1 1 1 1
1 0 0 1	1 0 0 1 0 0 1
0 1 0 1	0 1 0 1 0 1 0
1 1 0 1	1 1 0 1 1 0 0
0 0 1 1	0 0 1 1 1 0 0
1 0 1 1	1 0 1 1 0 1 0
0 1 1 1	0 1 1 1 0 0 1
1 1 1 1	1 1 1 1 1 1 1

Let us now introduce a single error into the code sequence, assuming that the sequence 1000011 is received. We could now compare the received sequence with every codeword as shown in Table 1.2. There is only one codeword that has a single difference from the received sequence, and that is our originally transmitted sequence 1100011. The systematic construction then makes it easy to extract the information 1100.

Table 1.2 Example of single-error correction

Received	Codeword	Distance
1 0 0 0 0 1 1	0 0 0 0 0 0 0	3
1 0 0 0 0 1 1	1 0 0 0 1 1 0	2
1 0 0 0 0 1 1	0 1 0 0 1 0 1	4
1 0 0 0 0 1 1	1 1 0 0 0 1 1	1
1 0 0 0 0 1 1	0 0 1 0 0 1 1	2
1 0 0 0 0 1 1	1 0 1 0 1 0 1	3
1 0 0 0 0 1 1	0 1 1 0 1 1 0	5
1 0 0 0 0 1 1	1 1 1 0 0 0 0	4
1 0 0 0 0 1 1	0 0 0 1 1 1 1	3
1 0 0 0 0 1 1	1 0 0 1 0 0 1	2
1 0 0 0 0 1 1	0 1 0 1 0 1 0	4
1 0 0 0 0 1 1	1 1 0 1 1 0 0	5
1 0 0 0 0 1 1	0 0 1 1 1 0 0	6
1 0 0 0 0 1 1	1 0 1 1 0 1 0	3
1 0 0 0 0 1 1	0 1 1 1 0 0 1	5
1 0 0 0 0 1 1	1 1 1 1 1 1 1	4

Of course the error created for the above example was in one of the information bits, so it might be as well to check what happens if the error falls in one of the calculated bits. Suppose we start again from the codeword 1100011 but assume that 1100111 is received. We repeat the previous exercise with the results shown in Table 1.3. Again we have chosen the correct codeword. Further examples may be tried and it will be found that single-bit errors are always recovered.

Table 1.3 Second example of single-error correction

Received	Codeword	Distance
1 1 0 0 1 1 1	0 0 0 0 0 0 0	5
1 1 0 0 1 1 1	1 0 0 0 1 1 0	2
1 1 0 0 1 1 1	0 1 0 0 1 0 1	2
1 1 0 0 1 1 1	1 1 0 0 0 1 1	1
1 1 0 0 1 1 1	0 0 1 0 0 1 1	4
1 1 0 0 1 1 1	1 0 1 0 1 0 1	3
1 1 0 0 1 1 1	0 1 1 0 1 1 0	3
1 1 0 0 1 1 1	1 1 1 0 0 0 0	4
1 1 0 0 1 1 1	0 0 0 1 1 1 1	3
1 1 0 0 1 1 1	1 0 0 1 0 0 1	4
1 1 0 0 1 1 1	0 1 0 1 0 1 0	4
1 1 0 0 1 1 1	1 1 0 1 1 0 0	3
1 1 0 0 1 1 1	0 0 1 1 1 0 0	6
1 1 0 0 1 1 1	1 0 1 1 0 1 0	5
1 1 0 0 1 1 1	0 1 1 1 0 0 1	5
1 1 0 0 1 1 1	1 1 1 1 1 1 1	2

Finally, let us see what happens if two errors occur, corrupting the transmitted sequence 1100011 to 1101001 (see Table 1.4). In this case we choose the wrong codeword 1101001 and decode the information as 1101. The decoder has chosen a codeword that differs in three places from the transmitted codeword, adding one error to the two occurring on the channel. Two of those errors have followed through into the information, but that is purely chance; in other cases just one or even all three of the decoding errors could have occurred in the information bits.

Table 1.4 Example of attempted double-error correction

Received	Codeword	Distance
1 1 0 1 0 0 1	0 0 0 0 0 0 0	4
1 1 0 1 0 0 1	1 0 0 0 1 1 0	5
1 1 0 1 0 0 1	0 1 0 0 1 0 1	3
1 1 0 1 0 0 1	1 1 0 0 0 1 1	2
1 1 0 1 0 0 1	0 0 1 0 0 1 1	5
1 1 0 1 0 0 1	1 0 1 0 1 0 1	4
1 1 0 1 0 0 1	0 1 1 0 1 1 0	6
1 1 0 1 0 0 1	1 1 1 0 0 0 0	3
1 1 0 1 0 0 1	0 0 0 1 1 1 1	4
1 1 0 1 0 0 1	1 0 0 1 0 0 1	1
1 1 0 1 0 0 1	0 1 0 1 0 1 0	3
1 1 0 1 0 0 1	1 1 0 1 1 0 0	2
1 1 0 1 0 0 1	0 0 1 1 1 0 0	5
1 1 0 1 0 0 1	1 0 1 1 0 1 0	4
1 1 0 1 0 0 1	0 1 1 1 0 0 1	2
1 1 0 1 0 0 1	1 1 1 1 1 1 1	3

By trying further examples, the reader should soon be convinced that this is a code which can be guaranteed to correct single errors, but invariably fails if two or more errors occur. It will be seen shortly that the guaranteed correction is fairly easily predicted from the properties of the code, but the invariable choice of the wrong codeword if extra errors occur is slightly unusual; most codes have cases where a received sequence may fall at equal distance from two or more codewords, leaving the decoder unable to choose. In such cases it would be usual to declare a detected error but not to attempt correction. For example, in the simple code shown in Table 1.5 it can be seen that any single error will be corrected and some double errors (e.g. changing 01011 to 00001) will be miscorrected. Some double errors (e.g. changing 01011 to 10011) will, however, be uncorrectable because the received sequence differs in two places from two or more codewords.

Table 1.5

Information	Code
0 0	0 0 0 0 0
0 1	0 1 0 1 1
1 0	1 0 1 0 1
1 1	1 1 1 1 0

1.6 Random error detection and correction capability of block codes

If we try to analyze the reasons for the error correction performance of the block code in the previous section, we reach the conclusion that starting from any codeword we would have to change at least three bits to create another codeword. This least distance measure between codewords is important in determining the properties of the code and is called the *minimum distance*, $d_{\min}$. If we change only one bit in a codeword, then we would have to change at least two more to reach another codeword, hence single-bit errors can be recovered. On the other hand two or more errors are likely (for this code certain) to leave us closer to another codeword than the original. We could instead aim merely to detect the presence of errors by detecting that the received sequence is not a codeword. In this case one or two errors must be detected, three or more errors might result in another codeword being received.

In general we can use block codes either for error detection alone, for error correction or for some combination of the two. Taking into account that we cannot correct an error that cannot be detected, we reach the following formula to determine the guaranteed error detection and correction properties, given the minimum distance of the code:

$$d_{\min} > s + t \tag{1.6}$$

where s is the number of errors to be detected and t ($\leqslant s$) is the number of errors to be corrected. Assuming that the sum of s and t will be the maximum possible, then

$$d_{\min} = s + t + 1$$

Thus if $d_{\min} = 5$, the possibilities are

$$s = 4 \qquad t = 0$$
$$s = 3 \qquad t = 1$$
$$s = 2 \qquad t = 2$$

5
Convolutional codes

5.1 Introduction

In Chapter 1 it was explained that codes for error control generally fell into two categories, namely block codes and convolutional codes. The subsequent three chapters were devoted to block codes for random error correction, and the time has now come to look at convolutional codes. The space devoted to convolutional codes will be rather less than that used for block codes, but this fact should not be taken as an indication of their relative importance. Indeed in some ways the reasons why it is difficult to say a lot about convolutional codes are the same as the reasons why convolutional codes should be given serious consideration for error control, particularly forward error correction on Gaussian channels.

The most obvious feature of convolutional codes in comparison with block codes is that they do not exhibit the same sort of algebraic complexity. Convolutional encoders are relatively simple and the codes do not usually fall into families in the same way as block codes. In addition there is one decoding method which is in a definable sense optimal, which is in principle applicable to all convolutional codes and which permits the use of soft-decision decoding with little increase in complexity. As a result this method, known as Viterbi decoding, is the usual choice for real-time applications. It is only the restrictions on code complexity and decoding speed that stop the Viterbi method from being ideal for all convolutional code applications.

A few years ago there appeared on British television an advertisement for a well-known brand of beer whose message could be taken as applying to convolutional codes. Two beer drinkers are shown in a bar. They order different beers and one of them proceeds to treat the other to a lecture on the merits of the former's chosen tipple. The second drinker says nothing, but finishes his drink whilst the first has barely started. The punch line is 'the less said, the better the beer.' As it happens, the beer was not one that I ever drink and neither am I a particular exponent of convolutional codes, nevertheless the point is well made. This is my excuse for the relative paucity of the treatment of convolutional codes.

As background to this chapter most of the first chapter is relevant, even

though the examples in Chapter 1 were based on a block code. Some of the characteristics of convolutional codes are presented as an analogy with the block code features, so the information on block codes will be needed. (This approach of block codes first, convolutional second is not necessary but it is convenient, and even authors whose personal bias is towards convolutional codes tend to follow it.) Because of the relevance of soft-decision decoding to convolutional codes, Sections 1.3 and 1.4 will be of interest. The discussion of linearity in Chapter 2 is also relevant because convolutional codes are linear, although the property that linearity implies the existence of an equivalent systematic code does not apply to convolutional codes.

5.2 General properties of convolutional codes

An example schematic diagram of a convolutional encoder is shown in Figure 5.1. Note that the information bits do not flow through directly into the code-stream, i.e. the code is not systematic. The difference between systematic and nonsystematic codes is not trivial with convolutional codes, and nonsystematic codes can give better performance when coupled with an appropriate decoding method.

The way in which the encoder works is that the input bit is modulo 2 added to stored values of previous input bits, as shown in Figure 5.1, to form the outputs which are buffered ready for transmission. The input bit is then moved into the shift registers and all the other bits shift to the left (the leftmost, i.e. oldest, stored bit being lost). The encoder starts with the registers clear, and if a 1 is input the outputs will be 11 and the encoder will contain 1 in the right-hand shift register stage. If the next input is 0 then the encoder will output 10, the right shift register will contain 0 and the left shift register 1.

There are few well-defined families of convolutional codes. The designer of the coding scheme usually chooses a decoding method and then chooses a suitable code for that method, the code having been selected by computer search techniques. Study of convolutional codes therefore tends to concentrate more on

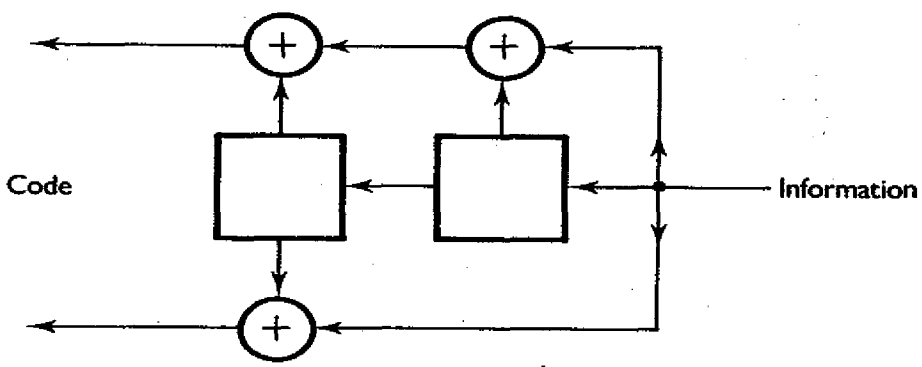

Figure 5.1 Convolutional encoder

decoding than on the codes themselves, but some appreciation of the encoding process will first be required.

5.3 Generator polynomials

The encoding operation could be described by two polynomials, one to represent the generation of the each output bit from the input bit. For the above code they are

$$g^{(1)}(X) = X^2 + X + 1$$
$$g^{(2)}(X) = X^2 + 1$$

The operator X represents a single frame delay.

The interpretation of these polynomials is that the first output bit is given by the modulo 2 sum of the bit that has been remembered for two frames (the X^2 term), the bit remembered for one frame (X) and the input bit (1). The second output is the bit remembered for two frames (X^2) modulo 2 added to the input (1).

The concept of generator polynomials can be applied also to cases where several bits are input at once. There would then be a generator to describe the way that each of the input bits and its previous values affected each of the outputs. For example, a code that had 2 bits in and 3 out would need six generators designated $g_1^{(1)}(X)$, $g_1^{(2)}(X)$, $g_1^{(3)}(X)$, $g_2^{(1)}(X)$, $g_2^{(2)}(X)$ and $g_2^{(3)}(X)$.

5.4 Terminology

Terminology is a particularly tricky and confusing part of convolutional codes. The major problem is a lack of uniformity in the terms used by different authors, with many instances where two people use different words for the same thing and, even worse, where two people use the same word for different things. As a result there cannot be said to be any consensus around the terms I have used and although I think I can find some support for each term, I cannot point to another author who uses the same terms for everything.

The terms to be used here to describe a convolutional code are as follows:

Input frame – the number of bits, k_0, taken into the encoder at once.

Output frame – the number of bits, n_0, output from the encoder at once.

Memory order – the maximum number, m, of shift register stages in the path to any output bit.

Memory constraint length – the total number, ν, of shift register stages in the encoder, excluding any buffering of input and output frames.

Input constraint length – the total number, K, of bits involved in the encoding operation; equal to $\nu + k_0$.

Output constraint length – the number, n, of output bits for which the effect of any one input bit persists; equal to $(m + 1)\, n_0$.

The term that causes particular problems is *constraint length*. Used without qualification, it might well mean what I have called *input constraint length*, but it could also be one of the two other possibilities or another related but differently defined parameter. Curiously enough, the symbol used (ν, K or n) is often more widely accepted than the term to describe it, and so may give a clue to the real meaning. The general message, however, is to be wary of the terms used by other authors and be sure to know what is meant.

A convolutional code may be termed a (n_0, k_0, m) code or a (n, k) convolutional code, where $k = (m + 1)k_0$. The latter stems from a philosophy that treats convolutional codes similarly to block codes and will not be followed here. Conversion from the latter to the former terminology can often be achieved by recognizing the common factor $m + 1$ between n and k. The term code rate, meaning $k_0 n_0$, is also often applied to convolutional codes.

For our example, $k_0 = 1$, $n_0 = 2$, $m = 2$, $\nu = 2$, $K = 3$ and $n = 6$. The code is a (2,1,2) convolutional code, the code rate being 1/2.

Convolutional codes are part of a larger family of codes called tree codes, which may be nonlinear and have infinite constraint length. If a tree code has finite constraint length, which means in practice that the encoder has no feedback, then it is a trellis code. A linear trellis code is a convolutional code.

5.5 Encoder state diagram

If an encoder has ν shift register stages, then the contents of those shift registers can take 2^ν states. The way in which the encoder transits between states will depend on the inputs presented in each frame. The number of possible input permutations to the encoder in a single frame is 2^{k_0}. Hence if $\nu > k_0$ not all states can be reached in the course of a single frame, with only certain states being connected by allowed transitions. The encoder states can be represented in diagrammatic form with arcs to show allowed transitions and the associated input and output frames, as in Figure 5.2 which shows the transitions for the encoder of Figure 5.1. The label given to each state indicates the contents of the encoder memory, and the bits in the input and output frames are listed in an arbitrary, but consistent, order. Since the input frame is represented in the end state number, it is not always necessary to include input values in the state diagram.

It is also fairly easy to work in reverse order and to derive the encoder

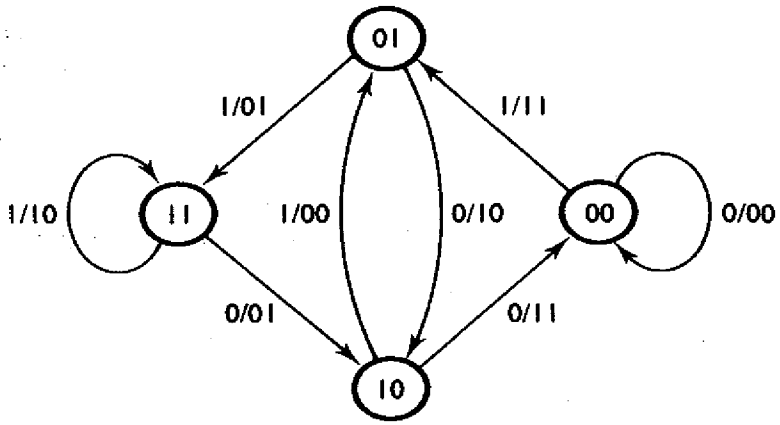

Figure 5.2 Encoder state diagram

circuit or the generator polynomials from the state diagram. The memory constraint length is easily determined from the number of states. The number of branches from each state (and the labels if inputs are shown) allows the size of the input frame to be determined. The shortest path from the zero state (memory clear) to the state where the memory is filled with ones gives the memory order. To find the generator polynomials describing the output contribution of a single bit of the input frame, we start with the encoder clear, inject one frame in which just that input bit is 1, then continue with a number of zero frames until the encoder is again clear. The first output frame will give the unity terms in the generator polynomials, the second output frame gives the X terms of the polynomials and so on.

For our example, the first output frame would be 11, indicating that the unity term is present in both polynomials. The second output frame of 10 indicates that the X term is present in $g^{(1)}(X)$ but not in $g^{(2)}(X)$. The third output frame of 11 indicates that both polynomials contain the X^2 term.

5.6 Distance structure of convolutional codes

As with block codes, there are concepts of distance that determine the error correcting power of the code. Because of linearity, one can assess the distance properties of the code relative to the all-zero path, but since the code sequence may be infinite in length, it is not at all clear how long the sequences to be compared should be. There are $m + 1$ frames involved in any encoding operation, which means that the minimum sensible path length for comparison would be $m + 1$ frames, i.e. a sequence of n bits. On the other hand, we might want to compare infinite sequences. The significance of these two possible approaches can be appreciated by consideration of the encoder state diagram of Figure 5.2.

We want to compare two paths which start from the same point and then diverge. Linearity means that we can use any path as the baseline for our comparison, and as in the block code case the all-zero sequence is convenient. We define a minimum distance as the weight of the minimum weight sequence of length $m + 1$ frames which deviates from the all-zero path. Thus we can see for the code in question that we must consider the possible paths of length 3, and we find that the minimum weight path follows the state sequence 00–01–10–01, giving $d_{\min} = 3$.

Now consider what happens if we extend the length of the paths we are comparing. The lowest weight path of length 4 follows the state sequence 00–01–10–01–10 and has weight 4. The lowest weight five-frame path is 00–01–10–01–10–01, again with weight 4. If we now move on to six-frame paths, a new path comes into contention, namely 00–01–10–00–00–00–00 which has weight 5, as does 00–01–10–01–10–01–10. This new path, however, has the property that it can be extended to infinite length without any increase in weight, merely by following the loop from 00 back to 00. Thus for a path of infinite length, the least value of Hamming distance is in this case 5. We call this value the free distance of the code and define it as the weight of the minimum weight sequence which deviates from the all-zero path and returns to it a number of frames later. The symbol used for free distance is either d_{free} or d_∞. The number of frames in the nonzero segment of the path used to calculate free distance is called the free length, n_{free}, of the code. In our example case, the free length is 6.

Of the two distances defined above, the one that is important in determining the error control properties of the code depends on the decoding method used. Some methods look at just one output constraint length at a time, treating the code almost as if it were a block code, and in that case it is $d_{\min}$ that is important. True maximum likelihood methods, however, compare complete received sequences with possible code sequences and for such methods d_∞ is the appropriate distance measure. In the example code, a free distance of 5 means that maximum likelihood decoding of any error pattern affecting not more than 2 bits would result in the original code sequence being recovered. Of course errors affecting more than 2 bits may be successfully decoded if the incorrect bits are sufficiently far apart.

It is in this context that the importance of nonsystematic codes arises, because they are needed to obtain values of free distance that are higher than $d_{\min}$. Odenwalder (1970) provides generators for rate 1/2 codes chosen for optimal output bit error rates at high signal-to-noise ratios, which is not necessarily the same as choosing the best value of free distance, although there is obviously a close link. These codes are shown in Table 5.1 with the generators shown as octal characters so that, for example, 15 represents the pattern 1101 or $X^3 + X^2 + 1$.

Table 5.1 Rate 1/2 convolutional codes

ν	$g^{(1)}, g^{(2)}$	d_∞
2	7,5	5
3	17,15	6
4	35,23	7
5	75,53	8
6	171,133	10
7	371,247	10
8	753,561	12

5.7 Evaluating distance and weight structure

In the previous section, the method used for finding the minimum weight paths of different lengths relied heavily on inspection and it would have been easy to miss one. If we confine ourselves to paths that start and end at the zero state, then there is a more formal method which finds the number of paths of any weight or length. The method is explained in the context of the previous example, based on the state diagram of Figure 5.2. The start point is to rearrange the state diagram so that the state 00 appears at each end of a network of paths, thus representing both the start and end points of the paths of interest. This is shown in Figure 5.3.

As the encoder moves from state to state, three things of possible interest happen. The length of the code sequence increases and the weights of both the input and output sequences either increase or remain the same. We define operators D, corresponding to an increase of 1 in the output weight: L, corresponding to an increase of one frame in the code sequence length; and N, representing an increase of 1 in the input sequence weight. We can now label each arc of the modified state diagram with the appropriate operators, as has been done on Figure 5.3; let X_i represent the accumulated weights and lengths

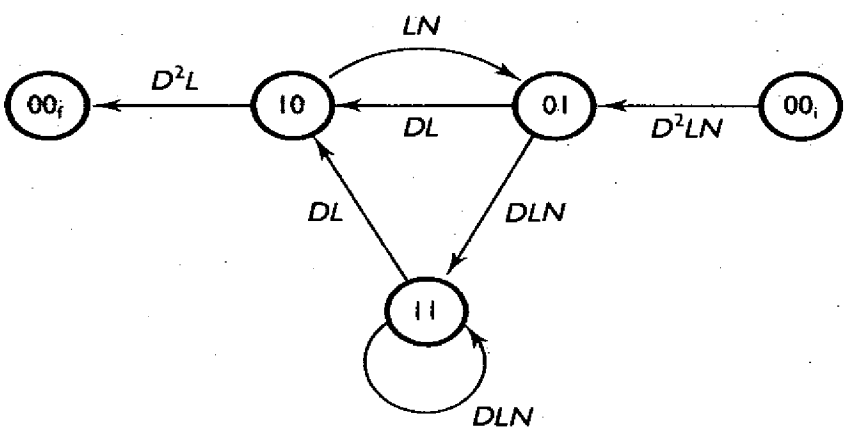

Figure 5.3 Modified encoder state diagram